ZZW AF
2654

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

OIPE JC101
APR 25 2005
PATENT & TRADEMARK OFFICE

In re Applicant:	John W. Merrill	§	Art Unit:	2654
Serial No.:	09/115,359	§	Examiner:	David D. Knepper
Filed:	July 14, 1998	§	Atty Docket:	ITL.0038US P5634
For:	Automatic Speech Recognition	§	Assignee:	Intel Corporation

Mail Stop **Appeal Briefs**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REPLY BRIEF

Sir:

This Reply Brief addresses the new points raised by the Examiner in his Answer.

1. Is Claim 33 Anticipated by Trower?

The claims were amended to attempt to make it clear that a single software object is used for both spoken and non-spoken inputs. The Applicant continues to believe that the term "object" has a well known meaning in the technical field. Given that one skilled in the art would only interpret "object" in the context of a software invention and when preceded by the word "software," as an object as defined in the various technical sources, there is no basis to interpret "object" in a non-technical way. The Examiner has also adopted the technical as opposed to the non-technical definition.

The following sources of definition can be identified:

First, the Microsoft Computer Dictionary unequivocally defines "object" (See Appendix A) in one and only one way and does not admit of any broader, popular definition of object to

Date of Deposit: April 20, 2005
I hereby certify under 37 CFR 1.8(a) that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage on the date indicated above and is addressed to: Mail Stop Appeal Brief-Patents, Commissioner for Patents, PO Box 1450, Alexandria, VA 22313-1450.
Cynthia L. Hayden
Cynthia L. Hayden

BEST AVAILABLE COPY

mean effectively anything. In other words, when the term was defined in the Microsoft Computer Dictionary it was never contemplated that, in the computer field, one would interpret "object" to m

ean anything other than the software object as defined therein. This is strong evidence that one skilled in the art would never understand the term "object" in a software context as meaning effectively anything.

Secondly, the McGraw Hill Computer Desktop Encyclopedia, like the Microsoft Computer dictionary, does not even consider the popular concept of an object, but, instead, defines merely the term object as being a software object of a specific type, namely, the type used in object oriented programming. See Appendix B.

Thirdly, a source book called ActiveX and OLE, by David Chappell, defines an "object," at pages 9 and 10, as having a few key characteristics, including a set of data called state or attributes and a group of methods. See Appendix C.

Fourthly, it is incontrovertible that the Examiner, himself, accepted such a definition. See e.g., Examiner's Answer at page 10, lines 14-17.[1] The Examiner, in his previous appeal brief at page 6, calls for "firing the same event when receiving spoken and non-spoken commands is clearly included by Trower, who will allow an object's methods to be controlled by more than one interface." If "object" simply means anything including a computer system or any type of software, where did the concept of methods come from? It is clear that the Examiner, during prosecution, recognized what an object was and applied that technical definition.

Fifthly, the cited Trower reference itself also specifically defines "object" as an instance that a programmer defined type referred to as a class, which exhibits the characteristics of data encapsulation etc. See column 18, lines 21-26.

In the prior appeal, on reconsideration, the Board refused to consider the technical definition of object, stating that it should have been raised during the prosecution. In the continuing application, this very argument was made and was considered. Now it is ripe for the Board's decision, as consistently conceded by the Examiner in this appeal.. In the prior decision at page 5, the Board never found that an object in the sense of an object oriented object was contained in Trower but, instead, simply pointed to Trower's entire computer system as

---

1 The Examiner also clarified that the Applicant's "software object" is not defined in the claims.

suggesting that an object, in the non-technical sense, handled both spoken and non-spoken commands.

Given the proper interpretation of "software object," the Examiner's rejection should be reversed. The proper interpretation is one which would be understood by one skilled in the art. The Federal Circuit has been clear that the Board must use the technical definition. See *In re Cortright,* 165 F.3d 1353, 49 U.S.P.Q.2d 1464 (Fed. Cir. 1999). Therefore, there is simply no way to maintain a definition which no one skilled in the art would ever entertain in this context. In *In re Cortright,* 49 U.S.P.Q.2d at 1468, the Federal Circuit noted three separate patents used the same definition. Here, we have multiple dictionaries, prior art, and treatises, as well as the Examiner himself, using that definition.

Even if there were some way to contend that a software object can be anything, the modifier "software" precludes the object from reading on the entire computer system. Claim 33 calls for providing a single software object. Thus, whatever the object is, it must be software, it must be only one object, and it must receive both spoken and non-spoken commands. An event must be fired when the object receives spoken commands and an event must be fired when the object receives non-spoken commands, such that the same object can handle both spoken and non-spoken commands. The firing of an event is object oriented language. As set forth in the treatise ActiveX and Ole, cited above, at page 10, an object in object oriented programming fires an event. There is no basis to read firing an event on a general object in the non-technical sense, since firing an event is a characteristic of objects in the technical or object oriented sense.

In his Answer, the Examiner contends that the Appellant has failed to indicate why the Board's decision that the entire computer is insufficient to meet the claimed invention. While it is true that the Board did find that the entire computer did do what was claimed, what is actually claimed more clearly now is that it must be software, it must be some software object, and it must be a single software object that fires an event for both spoken and non-spoken commands. There is simply no way to assert that the same software object in Trower fires an event for both spoken and non-spoken commands.

For example, in the context of a Section 103 rejection, the Federal Circuit found that a claim that called for a sensor to control a number of valves was not met by a system to control a number of valves. *In re Kotzab*, 217 F.3d 1365, 55 U.S.P.Q.2d 1313 (Fed. Cir. 2000). The Federal Circuit even noted that there was not substantial evidence to show that one system is the

same thing as one sensor. Similarly, here, what is being done is to argue that the entire system of Trower handles both spoken and non-spoken events despite the fact that the claim is specific that it must be a specified element of the system (a software object) which actually does the step. This is directly contrary to the facts and the law in the cited Kotzab case. Therefore, the rejection should be reversed.

Under paragraph 10 of the Answer, the Examiner contends that Trower teaches providing a single software object that receives spoken and non-spoken command information. In support thereof he cites language that says an agent object will respond to certain commands when a client becomes active, enabling its selection through speech recognition, citing column 27, lines 5-26. This discussion is about is a general utility which is not as specific as what is claimed. It is a utility that allows an object to be programmed to do certain things. But that text is not specific in that it does not explain that the same object receives both spoken and non-spoken commands. For example, that the client can also set a voice property for a command which enables selection through speech recognition does not indicate that the object would still receive and fire events in response to both spoken and non-spoken commands. It simply says it can respond to spoken commands if the voice property is set. It does not indicate that the same object also receives non-spoken commands even though the voice property was set. There is simply no teaching commensurate with the scope of the claims.

The second element of the claim discussed by the Examiner is firing an event when an object receives spoken command information. In support of the rejection, the Examiner cites the language in column 21, lines 17-23. But all this talks about is how standard Ole objects work. It does not in any way discuss an object which fires an event in response to a spoken command. The fact that a dual interface is referred to does not mean that the object responds to spoken and non-spoken commands. It simply says that programs can invoke an object's method through an interface directly through a second type of interface. There is no suggestion that the same object could handle both spoken and non-spoken commands. Again, the cited language is much more general than the specifics of the claim.

The fact that the Examiner observes that a single object can perform multiple functions does not teach that it can perform the multiple functions specifically claimed and there is no discussion whatsoever that the same object would respond to both spoken and non-spoken commands.

The Examiner relies on the summary of the invention and, specifically, column 2, lines 29-34. There, it is indicated that the clients can specify input commands including both speech and cursor input for a character. This language is not commensurate with the requirement that the same object handle both types of inputs. For example, an object may be programmed in the cited reference at one time to handle speech and one time to handle cursor, but it cannot ever receive and fire events for both input types it may receive. The difference is that then the user can provide an input through the cursor or through speech and the same object handles either input, in some embodiments, in the same way in the claimed invention. In the cited reference, the computer as a whole may operate on certain commands. The reference does not state that the same object can handle both spoken and non-spoken commands, even if it were true (and there is no support for such a conclusion) that the computer as a whole could handle both.

As pointed out above, once the property for voice is specified, there is no reason to believe that the same object could also handle non-voice inputs. It is reasonable to understand that specifying the voice property then dedicates that object to voice inputs. In such case, the same object cannot fire events when spoken commands, as well as non-spoken commands, are received.

Likewise, the material at the bottom of column 2 that the clients can specify speech or cursor input is noted, but that language does not mean that one can supply either input to the same object which will fire events in both cases. The fact that the server monitors inputs from the operating system is necessarily true if a given object can handle one, and only one, of the two input types. Since the system does not know what type the object is coded to handle, it must monitor for both inputs. Therefore, both citations are non-informing as to whether or not the same object handles both spoken and non-spoken commands. Moreover, there is no reason to believe that once the voice property of the object is programmed in, as described in the Trower reference, that the object can then also handle non-spoken inputs.

Therefore, the rejection based on Trower should be reversed.

II. Are Claims 33-52 Obvious Over Hashimoto Alone?

With respect to the rejection based on Hashimoto, it is respectfully submitted that there is no basis for a single reference Section 103 rejection. A *prima facie* rejection was not and cannot be made out. In order to support a *prima facie* rejection, there must be something in the

reference which teaches modifying itself. But if the reference teaches modifying itself, it would be a Section 102 rejection. Given that the office action concedes that something is not taught in Hashimoto, that thing that is missing cannot come from the prior art because no prior art in conjunction with Hashimoto is cited. It is plainly the Examiner's burden of proof to cite that thing and to set out a *prima facie* case. See M.P.E.P. § 2142. Given that a *prima facie* rejection has not and cannot be made out, the Appellant has no duty to further respond. The Appellant can and has simply stood pat on the grounds that no *prima facie* rejection is made out. Therefore, there is simply no legal way to sustain a single reference Section 103 rejection based on Hashimoto or anyone else.

On page 7 of the Answer, it is conceded that Hashimoto does not teach firing an event related to a command. It is then concluded that it would have been obvious to one skilled in the art to do so. In other words, it is argued that even though "Hashimoto does not describe specific actions to be employed" somehow it would be obvious to do so in the absence of any prior art teaching. The fact that Hashimoto teaches that his interface will limit recognition in order to avoid wasteful matching processing with respect to unnecessary vocabularies, clearly and admittedly, cannot teach the missing element. If it is missing, it is missing. Suggestions to the contrary that do not reach the scope of the claim are insufficient to meet a *prima facie* rejection.

Particular findings are required to support a *prima facie* rejection of obviousness over a single reference. See *In re Kotzab*, 55 U.S.P.Q.2d at 1317. "Broad conclusory statements standing alone are not evidence." *Id.* For example, the assertion that Hashimoto would do what is claimed because it is desirable (Answer, page 6) is plainly prohibited hindsight reasoning. *Kotzab*, 55 U.S.P.Q.2d at 1318.

Similarly, the fact that Hashimoto teaches avoiding wasteful speech matching processing by using only a selected vocabulary, does not prove that Hashimoto, therefore, foresaw every advantageous way to implement any invention. Again, the argument is hindsight reasoning and illogical. Because in a different situation Hashimoto avoided wasteful processing does not prove that Hashimoto recognized and taught every advantageous way to process events. In other words, teaching the use of limited vocabularies cannot possibly teach firing an event related to a command.

Therefore, a *prima facie* rejection is not made out.

There is no software object and there is no single software object. To read the claim on the system as a whole is improper, as specifically found in the *Kotzab* case cited above. Namely, it is improper to simply equate "system" with "single software object" and to read the claim onto the system as a whole as explicitly prohibited in the *Kotzab* case. Specifically, the attempt to equate the SIM 104 to a single software object is unjustified. The SIM 104 is a "system interface management system." See Hashimoto at column 59, line 26. *Kotzab* precludes trying to equate a system with a specific type of thing which might be part of the system.

In Hashimoto, there is no reason to believe that anywhere within his system 104 resides the claimed single software object in the technical sense. Moreover, referring to Figure 96, it is clear that the item 104 cannot possibly be a single software object. It includes interface management unit 141, program operation registration unit 142, and message conversion unit 143. A review of these elements in columns 59 and 60 gives no basis to believe that they constitute a single software object.

In other words, the reason that the Examiner conceded that Hashimoto does not teach firing event is because Hashimoto does not teach a single software object. To simply claim that one skilled in the art would do anything that is better fails to make out a *prima* facie rejection. There is nothing in Hashimoto that suggests using the claimed better thing: a single software object. The fact that the claimed invention is better does not mean that Hashimoto teaches the claimed invention just because he realized speech recognition is better when one uses less than all of the available English vocabulary.

Therefore, the rejection should be reversed.

Respectfully submitted,

Date: April 20, 2005

Timothy N. Trop, Reg. No. 28,994
TROP, PRUNER & HU, P.C.
8554 Katy Fwy, Ste 100
Houston, TX 77024-1805
713/468-8880 [Phone]
713/468-8883 [Facsimile]

Attorneys for Intel Corporation

Microsoft Press Computer Dictionary

Third Edition

Microsoft Press

object \ob´jekt\ *n.* **1.** Short for object code (machine-readable code). **2.** In object-oriented programming, a variable comprising both routines and data that is treated as a discrete entity. *See also* abstract data type, module (definition 1), object-oriented programming. **3.** In graphics, a distinct entity. For example, a bouncing ball might be an object in a graphics program.

object code \ob´jekt kōd`\ *n.* The code, generated by a compiler or an assembler, that was translated from the source code of a program. The term most commonly refers to machine code that can be directly executed by the system's central processing unit (CPU), but it can also be assembly language source code or a variation of machine code. *See also* central processing unit.

object computer \ob´jekt kəm-pyōō`tər\ *n.* The computer that is the target of a specific communications attempt.

object database \ob´jekt dā`tə-bās\ *n. See* object-oriented database.

Object Database Management Group \ob`jəkt dā`tə-bās man´əj-mənt grōōp`\ *n.* An organization that promotes standards for object databases and defines interfaces to object databases. *Acronym:* ODMG (O`D-M-G´). *See also* Object Management Group.

object file \ob´jekt fīl`\ *n.* A file containing object code, usually the output of a compiler or an assembler and the input for a linker. *See also* assembler, compiler (definition 2), linker, object code.

Objective-C \əb-jek`tiv-C´\ *n.* An object-oriented version of the C language developed in 1984 by Brad Cox. It is most widely known for being the standard development language for the NeXT operating system. *See also* object-oriented programming.

object linking and embedding \ob`jekt lēnk`ēng and em-bed´ēng\ *n. See* OLE.

Object Management Architecture \ob`jekt man`əj-mənt är´kə-tek-chur\ *n. See* OMA.

Object Management Group \ob`jekt man´əj-mənt grōōp`\ *n.* An international organization that endorses open standards for object-oriented applications. It also defines the Object Management Architecture (OMA), a standard object model for distributed environments. The Object Management Group was founded in 1989. *Acronym:* OMG (O`M-G´). *See also* object model (definition 3), OMA, open standard.

object model \ob´jekt mod`əl\ *n.* **1.** The structural foundation for an object-oriented language, such as C++. This foundation includes such principles as abstraction, concurrency, encapsulation, hierarchy, persistence, polymorphism, and typing. *See also* abstract data type, object (definition 2), object-oriented programming, polymorphism. **2.** The structural foundation for an object-oriented design. *See also* object-oriented design. **3.** The structural foundation for an object-oriented application.

object module \ob´jekt moj`ōōl, mod`yōōl\ *n.* In programming, the object-code (compiled) version of a source-code file that is usually a collection of routines and is ready to be linked with other object modules. *See also* linker, module (definition 1), object code.

object-oriented \ob´jekt-ōr`ē-en-təd\ *adj.* Of, pertaining to, or being a system or language that supports the use of objects. *See also* object (definition 2).

object-oriented analysis \ob`jekt-ōr-ē-ent-əd ə-nal´ə-sis\ *n.* A procedure that identifies the component objects and system requirements of a system or process that involves computers and describes how they interact to perform specific tasks. The reuse of existing solutions is an objective of this sort of analysis. Object-oriented analysis generally precedes object-oriented design or object-oriented programming when a new object-oriented computer

Computer Desktop Encyclopedia

Ninth Edition

Alan Freedman

Osborne/McGraw-Hill
New York Chicago San Francisco
Lisbon London Madrid Mexico City Milan
New Delhi San Juan Seoul Singapore Sydney Toronto

O

OA See *office automation.*

OADG (Open Architecture Development Group) An organization founded by IBM Japan in 1991 to promote PC standards in Japan. See *DOS/V.*

OAI (Open Application Interface) A computer to telephone interface that lets a computer control and customize PBX and ACD operations.

OASIS (Organization for the Advancement of Structured Information Standards, Billerica, MA, www.oasis-open.org) A membership organization that is involved in promoting public information standards including XML, SGML and CGM. OASIS sponsors seminars, conference panels, exhibits and other educational events.

object (1) A self-contained module of data and its associated processing. Objects are the software building blocks of object technology. See *object technology* and *object-oriented programming.*

(2) In a compound document, an independent block of data, text or graphics that was created by a separate application.

object-based Having to do with object technology. See *object technology.*

object browser A utility that provides a hierarchical view of the Java classes in a Java application. It typically comes in an integrated development environment (IDE). See *IDE.*

object bus The communications interface through which objects are located and accessed. An object bus is a high-level protocol which rides on top of a lower-level transport protocol. See *CORBA* and *DCOM.*

object code The machine language representation of programming source code. Object code is created by a compiler and is then turned into executable code by the link editor. This is an early term that has no relationship to object technology. See *executable code, machine language* and *object-oriented programming.*

object computer Same as *target computer.* This is an early term that has no relationship to object technology.

object database See *object-oriented database.*

Object Database Management Group (Object Database Management Group, Burnsville, MN, www.odmg.org) An organization founded in 1991 to promote standards for object databases. The ODMG standard adds programming extensions to C++ and Smalltalk for accessing an object-oriented database. It also includes a superset of SQL 92 Entry Level, the most widely supported version of SQL.

The Object Database Management Group (ODMG) defines an interface to the database, whereas the Object Management Group (OMG) defines an interface for using objects in a distributed environment. The ODMG object model complies with the core model of the Object Management Architecture (OMA) of the OMG. See *OMA* and *CORBA.*

STRATEGIC TECHNOLOGY SERIES

UNDERSTANDING

ActiveX™ AND OLE

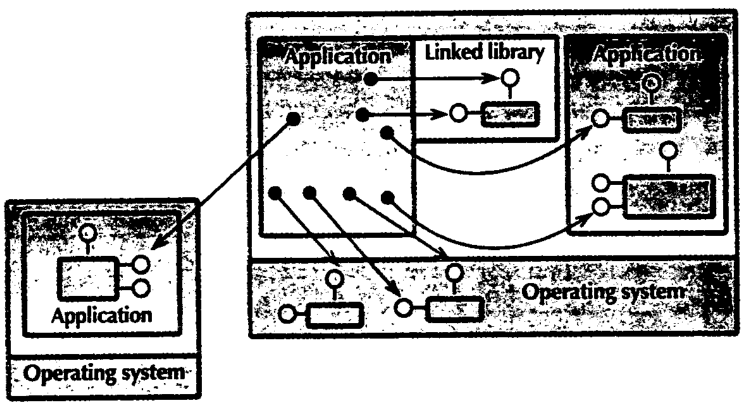
DAVID CHAPPELL

Microsoft Press

the methods in the interface. To a programmer, invoking a method looks like invoking a local procedure or function. In fact, however, the code that gets executed might be running in a library or in a separate process or as part of the operating system or even on another system entirely. With COM, clients don't need to be aware of these distinctions—everything is accessed in the same way. As shown in Figure 1-6, one common model is used to access services provided by all kinds of software.

Figure 1-6

With COM, an application accesses an object's services (no matter where that object resides) by invoking a method in an interface.



COM and Object Orientation

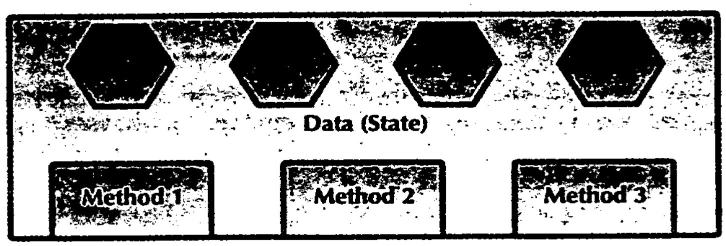
Objects are a central idea in COM. But how COM defines and uses objects sometimes differs from the way objects are used in other popular object technologies. To understand how COM relates to other object-oriented technologies, it's useful to describe what's commonly meant by the term *object-oriented* and then see how COM fits in.

An object is a combination of data and methods

Defining an object The term *object* has been blurred by marketeers trying to latch on to the latest fad, but in the minds of most, object-oriented technologies have a few key characteristics. Chief among these is a common notion of what constitutes an object. There is widespread agreement that an object consists of two elements: a defined set of data (also called *state* or *attributes*)

and a group of methods. These methods, commonly implemented as procedures or functions, allow a client of the object to ask the object to perform various tasks. Figure 1-7 shows a simple picture of an object.

Figure 1-7 ***An object has both methods and data.***



Unlike COM, most popular object technologies allow only a single interface per object

So far, so good—objects in COM are exactly like this. But in most object technologies, each object supports a single interface with a single set of methods. In contrast, COM objects can—and nearly always do—support more than one interface. An object in C++, for example, has only a single interface that includes all the object's methods. A COM object, with its multiple interfaces, might well be implemented using several C++ objects, one for each COM interface the object supports (although C++ isn't the only language that can be used to build COM objects).[2]

Another familiar idea in object technology is the notion of class. All objects representing bank accounts, for example, might be of the same class. Any particular bank account object, such as the one representing your account, is an *instance* of this class.

In COM, a class identifies a particular implementation of a set of interfaces

COM objects, too, have classes, as already described. In COM, a class identifies a specific implementation of a set of interfaces. Several different implementations of the same set of interfaces can exist, each of which is a different class. From the client's point of view, what matters are the interfaces. How those interfaces are implemented, which is what the class really indicates, isn't the

2 It's worth noting that, like COM objects, objects in the Java programming language can have multiple interfaces. In fact, as described in chapter 11, Java is a good fit for developing COM objects in several other ways, too.

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: ____________________

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.